Introduction to Software Reliability Risk Management

Norman F. Schneidewind

Norman F. Schneidewind, *PhD*
Naval Postgraduate School
2822 Racoon Trail
Pebble Beach, California 93953 USA
Internet (email): nschneid@nps.navy.mil

Summary and Purpose

This tutorial covers the following areas:

- Relationship between requirements attributes and reliability and maintainability

- Requirements attributes that are strongly related to the occurrence of defects and failures in the software

- Relationship between requirements attributes and software attributes like complexity and size

- Requirements attributes that are strongly related to the complexity and size of the software

- Feasibility of using requirements attributes as predictors of reliability and maintainability

- Static requirements change attributes, like the size of the change, for predicting reliability in execution (e.g., failure occurrence) and maintainability of the code

- Requirements attributes that pose the greatest risk for reliability and maintainability

This tutorial is designed for practitioner software engineers and managers who want to learn how to apply software reliability risk management in their organizations. Attendees will learn how to apply a software reliability risk analysis to identify fault prone software.

Norman F. Schneidewind, *PhD*

Norman F. Schneidewind is Professor of Information Sciences and Director of the Software Metrics Laboratory at the Naval Postgraduate School. Dr. Schneidewind is a Fellow of the IEEE, elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance". He is the developer of the Schneidewind software reliability model which is used by NASA to assist in the prediction of software reliability of the Space Shuttle, by the Naval Surface Warfare Center for Trident software reliability prediction, and by the Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction. This model is one of the models recommended by the American National Standards Institute and the American Institute of Aeronautics and Astronautics Recommended Practice for Software Reliability. He has published widely in the fields of software reliability and metrics.

Table of Contents

1. INTRODUCTION

While software design and code metrics have enjoyed some success as predictors of software quality attributes such as reliability [5, 6, 7, 8, 11, 13, 14], the measurement field is stuck at this level of achievement. If measurement is to advance to a higher level, we must shift our attention to the front-end of the development process, because it is during system conceptualization that errors in specifying requirements are inserted into the process and adversely affect our ability to maintain the software. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors propagate through later phases of development and maintenance. These errors may result in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements). Although requirements may be specified correctly in terms of meeting user expectations, there could be significant risks associated with their implementation. For example, correctly implementing user requirements could lead to excessive system size and complexity with adverse effects on reliability and maintainability or there could be a demand for project resources that exceeds the available funds, time, and personnel skills. Interestingly, there has been considerable discussion of project risk (e.g., the consequences of cost overrun and schedule slippage) in the literature [1] but not a corresponding attention to reliability and maintainability risk.

Risk in the Webster's New Universal Unabridged Dictionary is defined as "the chance of injury; damage, or loss" [21]. Some authors have extended the dictionary definition as follows: "Risk Exposure=Probability of an Unsatisfactory Outcome*Loss if the Outcome is Unsatisfactory" [1]. Such a definition is frequently applied to the risks in managing software projects such as budget and schedule slippage. In contrast, our application of the dictionary definition pertains to the risk of executing the software of a system where there is the chance of injury (e.g., crew injury or fatality), damage (e.g., destruction of the vehicle), or loss (e.g., loss of the mission) if a serious software failure occurs during a mission. We use risk factors to indicate the degree of risk associated with such an occurrence.

The generation of requirements is not a one-time activity. Indeed, changes to requirements can occur during maintenance. When new software is developed or existing software is changed in response to new and changed requirements, respectively, there is the potential to incur reliability and maintainability risks. Therefore, in assessing the effects of requirements on reliability and maintainability, we should deal with *changes* in requirements throughout the life cycle.

In addition to the relationship between requirements and reliability and maintainability there are the intermediate relationships between requirements and software metrics (e.g., size, complexity) and between metrics and reliability and maintainability. These relationships may interact to put the reliability and maintainability of the software at risk because the requirements changes may result in increases in the size and complexity of the software that may adversely affect reliability and maintainability. The Space Shuttle flight software is used as an example of these interactions. For example, the number of iterations of a requirements change -- the "modification level" – in the Space Shuttle is inversely related to reliability. That is, if many revisions of a requirement are necessary before it is approved, this is indicative of a requirement that is hard to understand and implement safely -- a risk that directly impacts reliability. At the same time, this complex requirement will affect the size and complexity of the code that will, in turn, have deleterious effects on reliability and maintainability.

Acronyms

CRs: Change Requests
DRs: Discrepancy Reports
MW: Man Weeks
SLOC: Source Lines Of Code

Assumption

There is a risk to reliability involved in making changes to the software.

Definitions

Alpha: Level of s-significance in statistical tests.

Categorical Data Analysis: A categorical variable is one for which the measurement scale consists of a set of categories.

Change History Line Count: Number of lines in the change record section of a module in an Operational Increment.

Change Request Count: Number of Change Requests per module in an Operational Increment.

Critical Value: A discriminant that distinguishes high quality from low quality software.

Cycle Count: Number of cycles per module in an Operational Increment.

Discrepancy Report Count: Number of Discrepancy Reports per module in an Operational Increment.

Discriminant: A factor that serves to classify entities into categories (e.g., software requirements complexity exceeds threshold, thus classifying the change as “high risk”).

Executable Statement Count: Number of source language statements that execute in a module of an Operational Increment.

Node Count: Number of nodes in a module's control graph in an Operational Increment.

Non-Commented Lines Of Code Count: Number of source lines of code, excluding comments, per module in an Operational Increment.

Operational Increment (OI): A software system comprised of modules and configured from a series of builds to meet Space Shuttle mission functional requirements.

Operand Count: Number of unique operands per module in an Operational Increment.

Operator Count: Number of unique operators per module in an Operational Increment.

Path Count: Number of paths per module in an Operational Increment.

Rank Correlation: A type of correlation analysis in which the ranks of quantities are used rather than their values.

Requirements Attributes: Characteristics of requirements (e.g., complexity, size, criticality).

s-significant: statistically significant.

SLOC Changed: Number of source lines of code changed per module in an Operational Increment.

Statement Count: Number of statements per module in an Operational Increment.

Severity Codes:

1. Severe Vehicle or Crew Performance Implications.
2. Affects Ability to Complete Mission (Not a safety issue).
3. Workaround Available, Minimal Effect on Procedures.
4. Insignificant (paperwork, etc.).
5. Not Visible to User.

2. OBJECTIVES

Given the lack of emphasis in software measurement on the critical role of requirements, it is important to explore the following issues:

- What is the relationship between requirements attributes and reliability and maintainability? That is, are there requirements attributes that are strongly related to the occurrence of defects and failures in the software?

- What is the relationship between requirements attributes and software attributes like complexity and size? That is, are there requirements attributes that are strongly related to the complexity and size of software?

- Is it feasible to use requirements attributes as predictors of reliability and maintainability? That is, can *static* requirements change attributes like the size of the change be used to predict reliability in *execution* (e.g., failure occurrence) and the maintainability of this code?

- Which requirements attributes pose the greatest risk to reliability and maintainability?

3. CONTEMPORARY SOFTWARE MEASUREMENT PROJECTS

A number of useful reliability and maintenance measurement projects have been reported in the literature. Much of the research and literature in software metrics concerns the measurement of code characteristics [10, 12]. This is satisfactory for evaluating product quality and process effectiveness once the code is written. However, if organizations use measurement plans that are limited to measuring code, these plans will be deficient in the following ways: incomplete, lack coverage (e.g., no requirements analysis and design), and start too late in the process. For a measurement plan to be effective, it must start with requirements and continue through to operation and maintenance. Since requirements characteristics directly affect code characteristics and hence reliability and maintainability, it is important to assess their impact when requirements are specified.

Briand, et al, developed a process to characterize software maintenance projects [2]. They present a qualitative and inductive methodology for performing objective project characterizations to identify maintenance problems and needs. This methodology aids in determining causal links between maintenance problems and flaws in the maintenance organization and process. Although the authors have related ineffective maintenance practices to organizational and process problems, they have not made a linkage to risk assessment.

Pearse and Oman applied a maintenance metrics index to measure the maintainability of C source code before and after maintenance activities [15]. This technique allowed the project engineers to track the "health" of the code as it was being maintained. Maintainability is assessed but not in terms of risk assessment.

Pigoski and Nelson collected and analyzed metrics on size, trouble reports, change proposals, staffing, and trouble report and change proposal completion times [17]. A major benefit of this project was the use of trends to identify the relationship between the productivity of the maintenance organization and staffing levels. Although productivity was addressed, risk assessment was not considered.

Sneed reengineered a client maintenance process to conform to the ANSI/IEEE Standard 1219, Standard for Software Maintenance [19]. This project is a good example of how a standard can provide a basic framework for a process and can be tailored to the characteristics of the project environment. Although applying a standard is an appropriate element of a good process, risk assessment was not addressed.

Stark collected and analyzed metrics in the categories of customer satisfaction, cost, and schedule with the objective of focusing management's attention on improvement areas and tracking improvements over time [20]. This approach aided management in deciding whether to include changes in the current release, with possible schedule slippage, or include the changes in the next release. However, the author did not relate these metrics to risk assessment.

An indication of the back seat that software risk assessment takes to hardware, Fragola reports on probabilistic risk management for the Space Shuttle. Interestingly, he says: "The shuttle ris k is embodied in the performance of its hardware, the careful preparation activities that its ground support staff take between flights to ensure this performance during a flight, and the procedural and management constraints in place to control their activities." [4]. There is not a word in this statement or in his article about software! Another hardware-only risk assessment is by Maggio, who says: "The current effort is the first integrated quantitative assessment of the risk of the loss of the shuttle vehicle from 3 seconds prior to liftoff to wheel-stop at mission end." Again, not a word about software [9].

Pfleeger lays out a roadmap for assessing project risk that includes risk prioritization [16], a step that we address with the degree of s-confidence in the statistical analysis of risk (see section 6).

4. TECHNICAL APPROACH

By retrospectively analyzing the relationship between requirements and reliability and maintainability, we will identify those risk factors that are associated with reliability and maintainability and we will prioritize them based on the degree to which the relationship is s-significant. In order to quantify the effect of a requirements change, we will use various risk factors that are defined as the attribute of a requirement change that can induce adverse effects on reliability (e.g., failure incidence), maintainability (e.g., size and complexity of the code), and project management (e.g. personnel resources). This process is illustrated in Figure 1. Various examples of Space Shuttle risk factors are shown in section 5.

Table 1 shows the Change Request Hierarchy of the Space Shuttle, involving change requests (i.e., a request for a new requirement or modification of an existing requirement), discrepancy reports (i.e., reports that document deviations between specified and observed software behavior), and failures. We will analyze categories 1 versus 2.1 and 1 versus 2.2.3 with respect to risk factors as discriminants of the categories.

Table 1: Change Request Hierarchy

Change Requests (CRs)

1. No Discrepancy Reports (i.e., CRs with no DRs)

2. Discrepancy Reports

    2.1 No failures (i.e., CRs with DRs only)

    2.2 Failures

        2.2.1 Pre-release failures

        2.2.2 Post-release failures

        2.2.3 Exclusive OR of 2.2.1 and 2.2.2 (i.e., CRs with failures)

4.1 Categorical Data Analysis

Using the null hypothesis, H_0: A risk factor is *not* a discriminant of reliability and maintainability versus the alternate hypothesis H_1: A risk factor is a discriminant of reliability and maintainability, we will use categorical data analysis to test the hypothesis. A similar hypothesis will be used to assess whether risk factors can serve as discriminants of metrics characteristics. We will use the requirements, requirements risk factors, reliability, and metrics data from the Space Shuttle "*Three Engine Out*" software (abort sequence invoked when three engines are lost) to test our hypotheses. Samples of these data are shown below.

- Pre-release and post release failure data from the Space Shuttle from 1983 to the present. An example of post-release failure data is shown in Table 2.

Table 2: Example Failure Data

Failure Found On Operational Increment	Days from Release When Failure Occurred	Discrepancy Report #	Severity	Failure Date	Release Date	Module in Error
Q	75	1	2	05-19-97	03-05-97	10

- Risk factors for the Space Shuttle *Three Engine Out Auto Contingency* software. This software was released to NASA by the developer on 10/18/95. An example of a partial set of risk factor data is shown in Table 3.

Table 3: Example Risk Factor Data

Change Request Number	SLOC Changed	Complexity Rating of Change	Criticality of Change	Number of Principal Functions Affected	Number of Modifications Of Change Request	Number of Requirements Issues	Number of Inspections Required	Manpower Required to Make Change
A	1933	4	3	27	7	238	12	209.3 MW

- Metrics data for 1400 Space Shuttle modules, each with 26 metrics. An example of a partial set of metric data is shown in Table 4.

Table 4: Example Metrics Data

Module	Operator Count	Operand Count	Statement Count	Path Count	Cycle Count	Discrepancy Report Count	Change Request Count
10	3895	1957	606	998	4	14	16

Table 5 shows the definition of the Change Request samples that are used in the analysis. Sample sizes are small due to the high reliability of the Space Shuttle. However, sample size is one of the parameters accounted for in the statistical tests that produced s-significant results in certain cases (see section 6).

Table 5: Definition of Samp les

Sample	**Size**
Total CRs	24
Instances of CRs with no DRs	12
Instances of CRs with DRs only	9
Instances of CRs with failures	7
Instances of CRs with modules that caused failures	7
CRs can have multiple instances of DRs, failures, and modules that caused failures. CR: Change Request. DR: Discrepancy Report.	

To minimize the confounding effects of a large number of variables that interact in some cases, a statistical categorical data analysis will be performed *incrementally*. We will use only one category of risk factor at a time to observe the effect of adding an additional risk factor on the ability to correctly classify change requests that have discrepancy reports (i.e., a report that documents deviations between specified and observed software behavior) or failures and those that do not. The Mann-Whitney test for difference in medians between categories will be used because no assumption need be made about s-statistical distribution; in addition, some risk factors are ordinal scale quantities (e.g., modification level), for which the median is an appropriate statistic. Furthermore, because some risk factors are ordinal scale quantities, rank correlation will be used to check for risk factor dependencies.

5. RISK FACTORS

One of the software maintenance problems of the NASA Space Shuttle Flight Software organization is to evaluate the risk of implementing requirements changes. These changes can affect the reliability and maintainability of the software. To assess the risk of change, the software development contractor uses a number of risk factors, which are described below. The risk factors were identified by agreement between NASA and the development contractor based on assumptions about the risk involved in making changes to the software. This formal process is called a risk assessment. No requirements change is approved by the change control board without an accompanying risk assessment. During risk assessment, the development contractor will attempt to answer such questions as: "Is this change highly complex relative to other software changes that have been made on the Space Shuttle?" If this were the case, a high-risk value would be assigned for the complexity criterion. To date this *qualitative* risk assessment has proven useful for identifying possible risky requirements changes or, conversely, providing assurance that there are no unacceptable risks in making a change. However, there has been no *quantitative* evaluation to determine whether, for example, high risk factor software was really less reliable and maintainable than low risk factor software. In addition, there is no model for predicting the reliability and maintainability of the software, if the change is implemented. We will address both of these issues.

Requirements attributes like completeness, consistency, correctness, etc. could be used as risk factors [3]. While these are useful generic concepts, they are difficult to quantify. Although some of the following risk factors also have qualitative values assigned, there are a number of quantitative risk factors, and many of the risk factors deal with the execution behavior of the software (i.e., reliability), which is of interest.

5.1 Space Shuttle Flight Software Requirements Change Risk Factors

The following are the definitions of the nineteen risk factors, where we have placed the risk factors into categories and have provided interpretations of the question the risk factor is designed to answer. If the answer to a yes/no question is "yes" or if the answer to a question that requires an estimate is an anomalous value, it means this is a high-risk change with respect to the given risk factor.

For each risk factor, it is indicated whether there is an s-significant relationship between it and reliability and maintainability for the software version analyzed. The details of the findings are shown in the section 6. In many instances, there is insufficient data to do the analysis because in these cases the risk factor evaluation forms are incomplete. These cases are indicated below. Only those risk factors where there is sufficient data (i.e. data from seven or more CRs) and the results are s-significant are shown. The names of the risk factors used in the analysis are given in quotation marks.

Complexity Factors

o Qualitative assessment of complexity of change (e.g., very complex); “complexity”. **Not s-significant**.

- Is this change highly complex relative to other software changes that have been made on the Space Shuttle?

o Number of modifications or iterations on the proposed change; “mods”. **s-significant.**

-How many times must the change be modified or presented to the change control board before it is approved?

Size Factors

o Number of lines of code affected by the change; “sloc”. **s-significant.**

- How many lines of code must be changed to implement the change request?

o Number of modules changed; “mod chg”. **Not s-significant.**

- Is the number of changes to modules excessive?

Criticality of Change Factors

o Criticality of function added or changed by the change request; “crit func” (**insufficient data**)

- Is the added or changed functionality critical to mission success?

o Whether the software change is on a nominal or off-nominal program path (i.e., exception condition); “off nom path”. (**insufficient data**)

- Will a change to an off-nominal program path affect the reliability of the software?

Locality of Change Factors

o The area of the program affected (i.e., critical area such as code for a mission abort sequence); “critic area” (**insufficient data**)

- Will the change affect an area of the code that is critical to mission success?

o Recent changes to the code in the area affected by the requirements change; “recent chgs” (**insufficient data**)

- Will successive changes to the code in one area lead to non-maintainable code?

o New or existing code that is affected; “new\exist code” (**insufficient data**)

- Will a change to new code (i.e., a change on top of a change) lead to non-maintainable code?

o Number of system or hardware failures that would have to occur before the code that implements the requirement would be executed; “fails ex code” (**insufficient data**)

- Will the change be on a path where only a small number of system or hardware failures would have to occur before the changed code is executed ?

Requirements Issues and Functions Factors

o Number and types of other requirements affected by the given requirement change (requirements issues); “other chgs” (**insufficient data**)

- Are there other requirements that are going to be affected by this change? If so, these requirements will have to be resolved before implementing the given requirement.

o Number of possible conflicts among requirements (requirements issues); “issues” **s-significant.**

- Will this change conflict with other requirements changes (e.g., lead to conflicting operational scenarios) o Number of principal software functions affected by the change; “prin funcs” **Not s-significant.**

- How many major software functions will have to be changed to make the given change?

Performance Factors

o Amount of memory space required to implement the change; “space” s-**significant.**

- Will the change use memory to the extent that other functions will not have sufficient memory to operate effectively?

o Effect on CPU performance; “cpu” (**insufficient data**)

- Will the change use CPU cycles to the extent that other functions will not have sufficient CPU capacity to operate effectively?

Personnel Resources Factors

o Number of inspections required to approve the change; “inspects” **Not s-significant.**

- Will the number of requirements inspections lead to excessive use of personnel resources?

o Manpower required to implement the change; “manpower”. **Not s-significant.**

- Will the manpower required to implement the software change be significant?

o Manpower required to verify and validate the correctness of the change; “cost” **Not s-significant.**

- Will the manpower required to verify and validate the software change be significant?

o Number of tests required to verify and validate the correctness of the change; “tests” **Not s-significant.**

- Will the number of tests required to verify and validate the software change be significant?

6. RESULTS

This section contains the results of performing the following statistical analyses (a, b, and c) shown in Tables 6. 7, and 8, respectively. This process is illustrated in Figure 2. Only those risk factors where there is sufficient data and the results are s-significant, as indicated in section 5, are shown. Some quantitative risk factors (e.g., size of change) are s- significant; no non-quantitative risk factors (e.g., complexity) are s-significant.

a. Categorical data analysis on the relationship between *CRs with no DRs* vs. *CRs with failures*, using the Mann-Whitney Test; and categorical data analysis on the relationship between *CRs with no DRs* vs. *CRs with DRs only*, using the Mann-Whitney Test

b. Dependency check on risk factors, using rank correlation coefficients; and

c. Identification of modules that caused failures as a result of the CR, and their metric values.

6.1Categorical Data Analysis

Of the original nineteen risk factors, only four survived as being s-significant (alpha ≤ .05); seven are not s-significant; and eight had insufficient data to make the analysis (see section 5). As Table 6 shows, there are s-significant results for *CRs with no DRs* vs. *CRs with failures* for the risk factors “mods”, “sloc”, “issues”, and “space”. There are also s-significant results for *CRs with no DRs* vs. *CRs with DRs only* for the risk factors “issues” and “space”. Since the value of alpha represents the level of s-significance of a risk factor in predicting reliability, we use it in Table 6 as a means to prioritize the use of risk factors, with low values meaning high priority. The priority order is: “space”, “issues”, “mods”, and “sloc”.

The s-significant risk factors would be used to predict reliability and maintainability problems for *this set of data and this version of the software*. The result regarding “mods” does confirm the software developer’s view that this is an important risk factor. This is the case because if there are many iterations of the change request, it implies that it is complex and difficult to understand. Therefore, the change is likely to lead to reliability and maintainability problems. It is not surprising that the size of the change “sloc” is s-significant because our previous studies of Space Shuttle metrics have shown it to be important [18]. Conflicting requirements “issues” could result in reliability and maintainability problems when the change is implemented. The on-board computer memory required to implement the change “space” is critical to reliability because unlike commercial systems, the Space Shuttle does not have the luxury of large physical memory, virtual memory, and disk memory to hold its programs and data. Any increased requirement on its small memory to implement a change comes at the price of demands from competing functions.

In addition to identifying predictive risk factors, we must also identify thresholds for predicting when the number of failures would become excessive (i.e., rise rapidly with the risk factor). An example is shown in Figure 3 where cumulative failures are plotted against cumulative memory space. The figure shows that when “space” reaches 2688 words, failures reach 3 (obtained by querying the data point) and climbs rapidly thereafter. Thus, a space count of 2688 would be the best estimate of the threshold to use in controlling the quality of the next version of the software. Similarly, Figure 4 shows that when “issues” reach a threshold of 286, failures reach 3 and rise rapidly thereafter. In contrast, Figure 5 shows no dramatic increase in failures with increase in “mods”; the reason for this anomaly is that there is less variability in the "mods" data than for the other three s-significant risk factors. Figure 6 shows that failures increase steeply when "sloc" reaches 1965.This process would be repeated across versions with the threshold being updated as more data is gathered. Thresholds would be identified for each risk factor in Table 6. This would provide multiple alerts for the quality of the software going bad (i.e., the reliability and maintainability of the software would degrade as the number of alerts increases).

6.2 Dependency Check on Risk Factors

In order to check for possible dependencies among risk factors that could confound the results, rank correlation coefficients are computed in Table 7. Using an arbitrary threshold of ≥.7, the results indicate s-significant dependencies between "issues" and "mod" and between "issues" and "sloc" for *CRs with no DRs*. That is, as the number of conflicting requirements increases, the number of modifications and size of the change request increases. In addition, there is an s-significant dependency between "issues" and "space" for *CRs with failures*. That is, as the number of conflicting requirements increases, the memory space required to implement the change request increases.

6.3 Identification of Modules that Caused Failures

Requirements change requests may occur on modules with metric values that exceed the critical values. In these cases, there is s-significant risk in making the change because such modules could fail. Table 8 shows modules that caused failures, as the result of the CRs, had metric values that far exceed the critical values. The latter were computed in [18]. A critical value is a discriminant that distinguishes high quality from low quality software. A module with metric values exceeding the critical values is predicted to cause failures.

Although the sample sizes are small, due to the high reliability of the Space Shuttle, the results consistently show that modules with excessive size and complexity lead to failures. Not only will the reliability be low but this software will also be difficult to maintain. The application of this information is that there is a high degree of risk when changes are made to software that has the metric characteristics shown in the table. Thus, these characteristics should be considered when making the risk analysis.

Table 6: S-significant Results (alpha ≤ .05). *CRs with no DRs* vs. *CRs. with failures*. Mann-Whitney Test

Risk Factor	**Alpha**	Median Value CRs with no DRs	**Median Value** CRs with failures
mods	.0168	.50	4
sloc	.0185	10	100
issues	.0038	2	16
space	.0036	4	231.5

CRs with no DRs vs. *CRs with DRs only*.

Risk Factor	**Alpha**	Median Value CRs with no DRs	Median Value CRs with DRs only
issues	.0386	2	14
space	.0318	4	111.50

mods: Number of modifications of the proposed change.
sloc: Number of lines of code affected by the change.
issues: Number of possible conflicts among requirements.
space: Amount of memory space required to implement the change (full words).

Table 7: Rank Correlation Coefficients of Risk Factors

	CRs with no DRs			
	mods	sloc	issues	space
mods		.230	**.791**	.401
sloc	.230		**.708**	.317
issues	**.791**	**.708**		.195
space	.401	.317	.195	
	CRs with failures			
	mods	sloc	issues	space
mods		.543	-.150	.378
sloc	.543		.286	.452
issues	-.150	.286		**.886**
space	.378	.452	**.886**	

Table 8: Selected Risk Factor Module Characteristics

Change Request	**Module**	**Metric**	**Metric Critical Value**	***Metric Value***
A	1	change history line count in module listing	63	558
A	2	non-commented lines of code count	29	408
B	3	executable statement count	27	419
C	4	unique operand count	45	83
D	5	unique operator count	9	33
E	6	node count (in control graph)	17	66

All of the above metrics exceeded the critical values for all of the above Change Requests.

7. CONCLUSIONS

Risk factors that are s-significant can be used to make decisions about the risk of making changes. These changes impact the reliability and maintainability of the software. Risk factors that are not s-significant should not be used; they do not provide useful information for decision-making and cost money and time to collect and process. The amount of memory space required to implement the change ("space"), the number of requirements issues ("issues"), the number of modifications ("mods"), and the size of the change ("sloc"), were found to be s-significant, in that priority order. In view of the dependencies among these risk factors, "space" would be the choice if the using organization could only afford a single risk factor. We also showed how risk factor thresholds are determined for controlling the quality of the next version of the software.

S-significant results were found for *CRs with no DRs* vs. *CRs with failures*; in addition, s-significant results were found for *CRs with no DRs* vs. *CRs with DRs only*.

Metric characteristics of modules should be considered when making the risk analysis because metric values that exceed the critical values are likely to result in unreliable and non-maintainable software.

This *methodology* can be generalized to other risk assessment domains, but the specific risk factors, their numerical values, and statistical results may vary.

8. REFERENCES

1] Barry W. Boehm, "Software Risk Management: Principles and Practices", IEEE Software, Vol. 8, No. 1, January 1991, pp. 32-41.

[2] Lionel C. Briand, Victor R. Basili, and Yong-Mi Kim, "Change Analysis Process to Characterize Software Maintenance Projects", Proceedings of the International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, pp. 38-49.

[3] Alan Davis, Software Requirements: Analysis and Specifications, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[4] Joseph R. Fragola, "Space Shuttle Program Risk Management", Proceedings Annual Reliability and Maintainability Symposium, 1996, pp. 133-142.

[5] Taghi M. Khoshgoftaar and Edward B. Allen, "Predicting the Order of Fault-Fault-Prone Modules in Legacy Software", Proceedings of the Ninth International Symposium on Software Reliability Engineering, November 4-7, 1998, Paderborn, Germany, pp. 344-353.

[6] Taghi M. Khoshgoftaar, Edward B. Allen, Robert Halstead, and Gary P. Trio, "Detection of Fault-Prone Software Modules During a Spiral Life Cycle", Proceedings of the International Conference on Software Maintenance, November 4-8, 1996, Monterey, California, pp. 69-76.

[7] Taghi M. Khoshgoftaar, Edward B. Allen, Kalai Kalaichelvan, and Nishith Goel, "Early Quality Prediction: A Case Study in Telecommunications", IEEE Software, Vol. 13, No. 1, January 1996, pp. 65-71.

[8] D. Lanning and T. Khoshgoftaar, "The Impact of Software Enhancement on Software Reliability", IEEE Transactions on Reliability, Vol. 44, No. 4, December 1995, pp. 677-682.

[9] Gaspare Maggio, "Space Shuttle Probabilistic Risk Assessment Methodology and Application", Proceedings Annual Reliability and Maintainability Symposium, 1996, pp. 121-132.

[10] Sebastian G. Elbaum and John C. Munson, "Getting a Handle on the Fault Injection Process: Validation of Measurement Tools", Proceedings of the Fifth International Software Metrics Symposium, November 20-21, 1998, Bethesda, Maryland, pp. 133-141.

[11] John C. Munson and Darrell S. Werries, "Measuring Software Evolution", Proceedings of the Third International Software Metrics Symposium, March 25-26, 1996, Berlin, Germany, pp. 41-51.

[12] Allen P. Nikora, Norman F. Schneidewind, and John C. Munson, IV&V Issues in Achieving High Reliability and Safety in Critical Control Software, Final Report, Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, California, January 19, 1998.

[13] Magnus C. Ohlsson and Claes Wohlin, "Identification of Green, Yellow, and Red Legacy Components", Proceedings of the International Conference on Software Maintenance, November 16-20, 1998, Bethesda, Maryland, pp. 6-15.

[14] Niclas Ohlsson and Hans Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches", IEEE Transactions on Software Engineering, Vol. 22, No. 12, December 1996, pp. 886-894.

[15] Troy Pearse and Paul Oman, "Maintainability Measurements on Industrial Source Code Maintenance Activities", Proceedings of the International Conference on Software Maintenance, Opio (Nice), France, October 17-20, 1995, pp. 295-303.

[16] Shari Lawrence Pfleeger, "Assessing Project Risk", Software Tech News, DoD Data Analysis Center for Software, vol.2, no. 2, pp. 5-8.

[17] Thomas M. Pigoski and Lauren E. Nelson, "Software Maintenance Metrics: A Case Study", Proceedings of the International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, pp. 392-401.

[18] Norman F. Schneidewind, "Software quality control and prediction model for maintenance", Annals of Software Engineering, Baltzer Science Publishers, Volume 9 (2000), May 2000, pp. 79-101.

[19] Harry Sneed, "Modelling the Maintenance Process at Zurich Life Insurance", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 217-226.

[20] George E. Stark, "Measurements for Managing Software Maintenance", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 152-161.

[21] Webster's New Universal Unabridged Dictionary, Second Edition, Simon and Shuster, New York, 1979.

9. BIBLIOGRAPY

1. *Recommended Practice for Software Reliability*, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

2. C. Billings, et al, "Journey to a Mature Software Process", *IBM Systems Journal*, Vol. 33, No. 1, 1994, pp. 46-61.

3. W. Farr and O. Smith, *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide*, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

4. Capers Jones, Capers, *Assessment and Control of Software Risks*, Yourdon Press, Prentice Hall, Upper Saddle River, NJ, 1994.

5. T. Keller and N. Schneidewind, A Successful Application of Software Reliability Engineering for the NASA Space Shuttle, *Software Reliability Engineering Case Studies*, International Symposium on Software Reliability Engineering, November 3, Albuquerque, New Mexico, November 4, 1997, pp. 71-82.

6. T. Keller, N. Schneidewind, and P. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", *Proceedings of the AIAA Computing in Aerospace 10*, San Antonio, TX, March 28, 1995, pp. 1-8.

7. Dale Walter Karolak, *Software Engineering Risk Management*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

8. N. Leveson, "Software Safety: What, Why, and How", *ACM Computing Surveys*, Vol. 18, No. 2, June 1986, pp. 125-163.

9. M. Lyu (Editor-in-Chief), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA and McGraw-Hill, New York, NY, 1995.

10. J. Musa, et al, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

11. A. Nikora, N. Schneidewind, and J. Munson, "Practical Issues In Estimating Fault Content And Location In Software Systems", *Proceedings of the AIAA Space Technology Conference and Exposition*, Albuquerque, NM, Sep 29-30, 1999.

12. A. Nikora, N. Schneidewind, and J. Munson, "IV&V Issues in Achieving High Reliability and Safety in Critical Control System Software" *Proceedings of the Third International Society of Science and Applied Technologies Conference on Quality in Design*, Anaheim, California, March 12-14, 1997, pp. 25-30.

13. N. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics", *IEEE Transactions on Software Engineering*, Vol. 25, No. 6, November/December 1999, pp. 768-781.

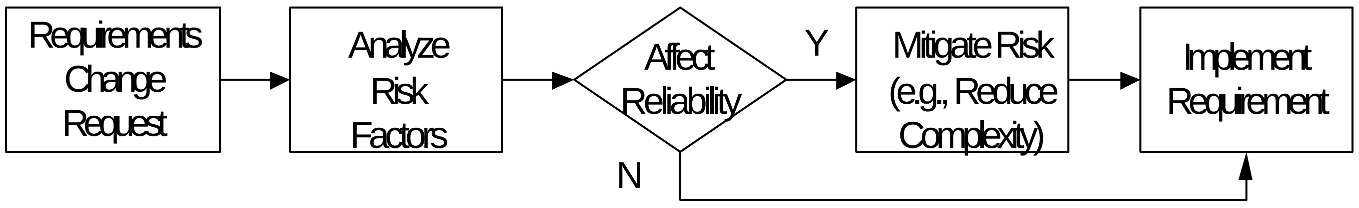
14. N. Schneidewind, "Software Validation for Reliability", *Wiley Encyclopedia of Electrical and Electronics Engineering*, John G. Webster, editor, John Wiley & Sons, Inc., Vol.19, 1999, pp. 607-618.

15. N. Schneidewind, "Reliability Modeling for Safety Critical Software", *IEEE Transactions on Reliability*, Vol. 46, No.1, March 1997, pp.88-98.

16. N. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", *IEEE Transactions on Software Engineering*, Vol. 19, No. 11, November 1993, pp. 1095-1104.

17. N. Schneidewind and T. Keller, "Application of Reliability Models to the Space Shuttle", *IEEE Software*, Vol. 9, No. 4, July 1992 pp. 28-33.

Figure 1: Risk Analysis Process


Requirements Change Request
Analyze Risk Factors
Affect Reliability
Y
N
Mitigate Risk (e.g., Reduce Complexity)
Implement Requirement

Check for S-Significant Relationships between Risk Factors and Failure Occurrence and Metric Characteristics

Figure 2: S-Significant Risk Factors

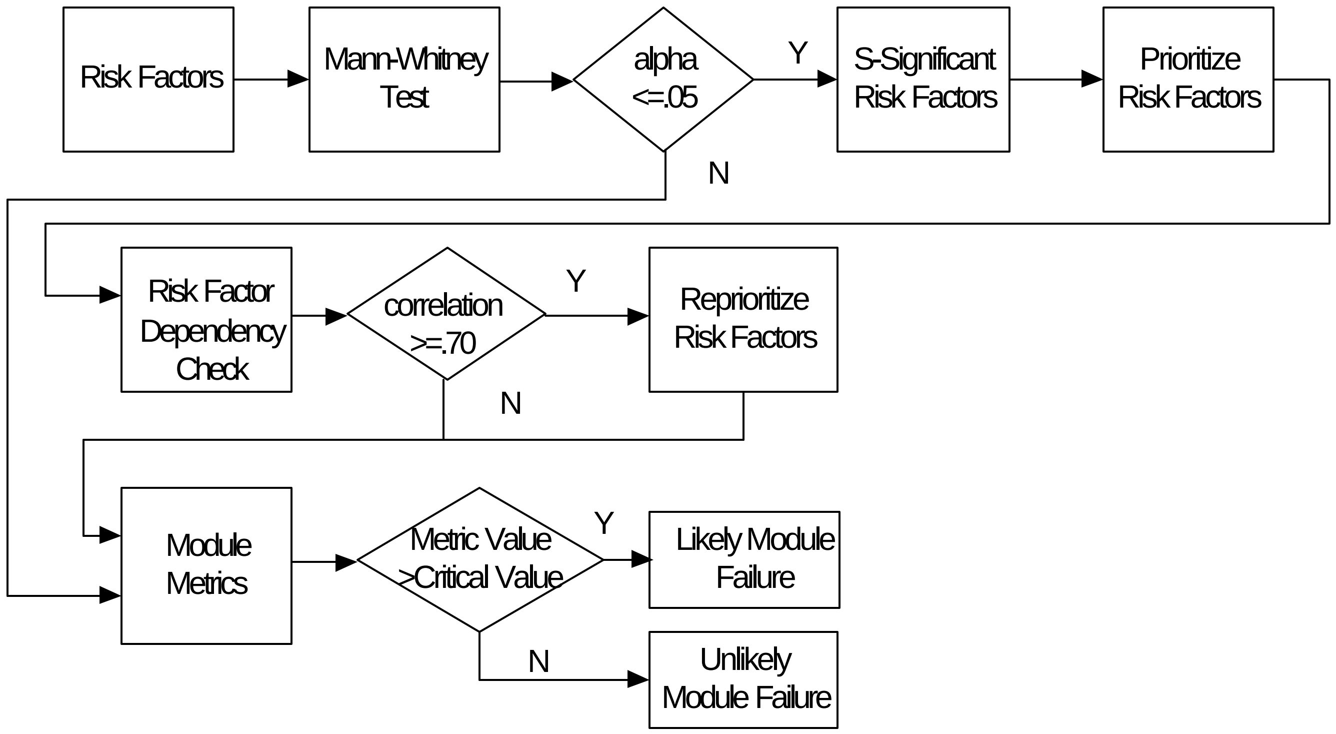


Figure 3: Failures vs. Memory Space

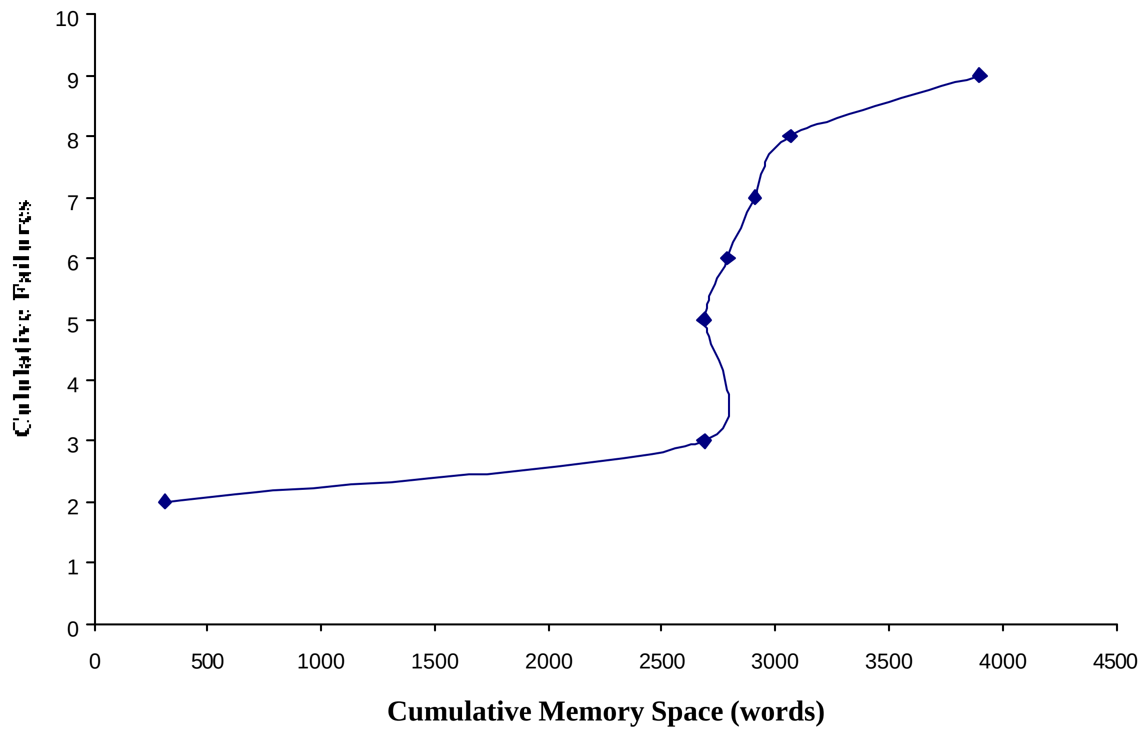


Figure 4: Failures vs. Issues

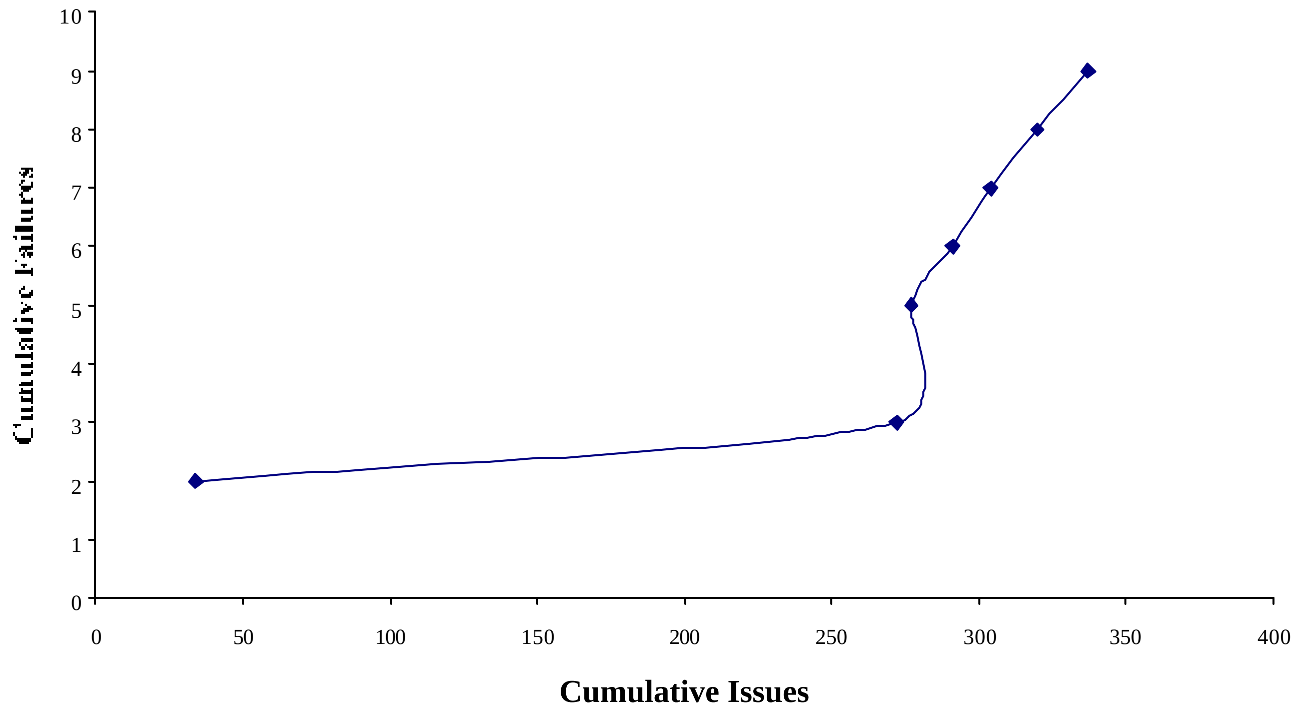


Figure 5: Failures vs. Mods

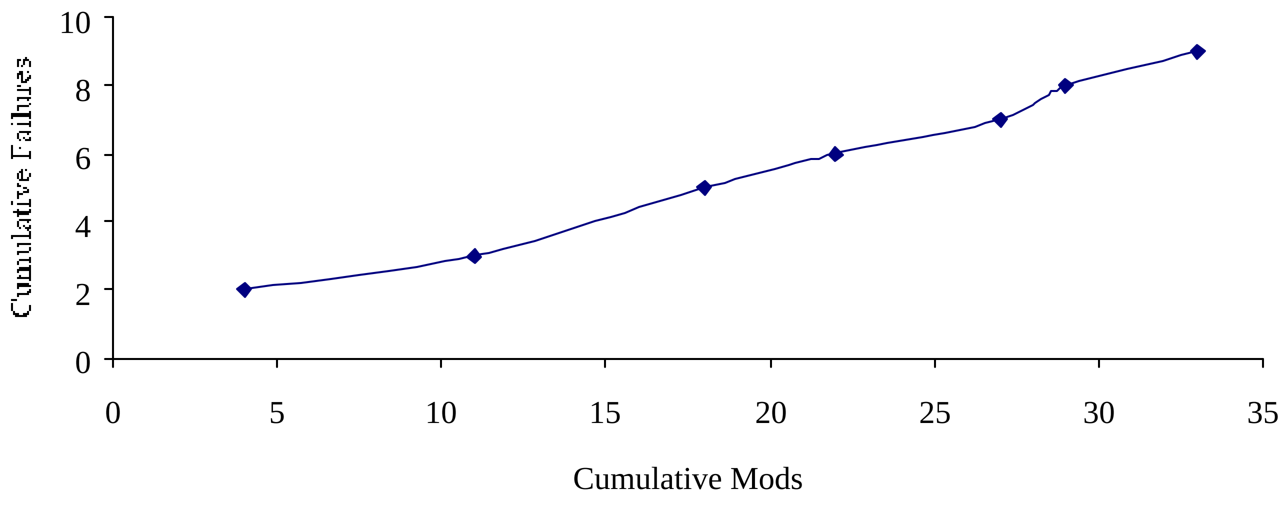
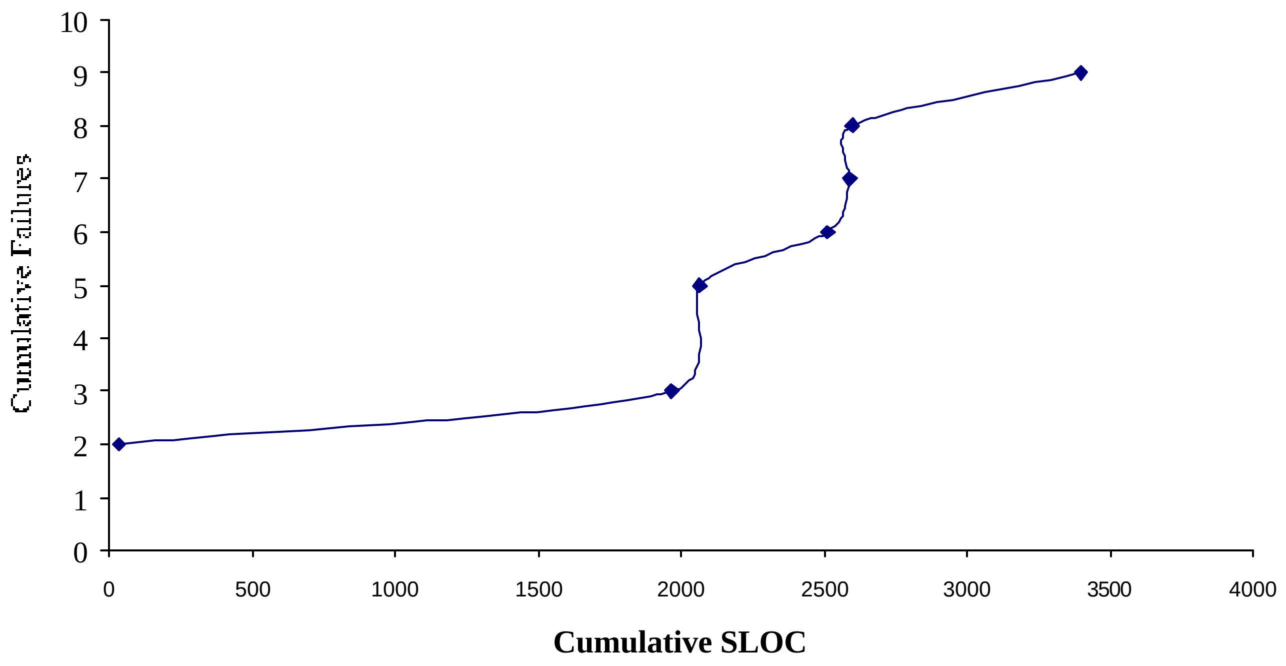


Figure 6: Failures vs. SLOC


Cumulative Failures
Cumulative SLOC